بسم الله الرحمن الرحيم

اساسيات البرمجة بلغة الجافا

الجزء الاول

محمد محمود ابراهيم

جامعة الزعيم الازهري

كلية علوم الحاسوب وتقانة المعلومات

اهداء : ـــ

الي طلاب كلية علوم الحاسوب وتقانة المعلومات جامعة الزعيم الازهري الي الدفعة 16 الي كل من ساهم في وصول هذا العمل الي هذا الشكل وارجوا ان ينال رضاءكم .

مقدمة : –

تعتبر لغة جافا من اللغات الحديثة جداً في عالم البرمجة، حيث ظهرت بصورة رسمية عام 1990م وقد قامت بوضع مفاهيمها شركة **Sun Microsystems** . وكان الغرض من ابتكار هذه اللغة برمجة صفحات الإنترنت. انتشرت لغة جافا حول العالم بسرعة كبيرة مع انتشار برمجة صفحات الإنترنت وبرمجة التطبيقات الحديثة الأخرى التي توفرها اللغة مثل برمجة شرائح الهاتف المحمول والبيجر والحواسيب الدفترية وغيرها.

مميزات لغة الجافا : –

- إنها لغة قوية تحتوي على أدوات كثيرة تساعد في كتابة البرامج.
- لكون جافا لغة حديثة مكنها من تلافي عيوب كثير من اللغات قبلها، من أهم هذه العيوب إمكانية الوصول المباشر لمواقع الذاكرة الخاصة بالبرنامج والذي يؤدي إلى ضعف سرية المعلومات وسهولة تدميرها.
- إن البرنامج المكتوب بلغة جافا يمكن نقله وتشغيله على جهاز حاسوب آخر يحتوي على نظام تشغيل يختلف عن الحاسوب الأول (مثلاً يحتوي **Windows, Linux** وغيرهما) بدون مشاكل.

- تعتبر لغة جافا لغة برمجة بالكائنات (**Object Oriented Programming Language**) ، ويعتبر هذا الصنف من لغات البرمجة من أوسعها انتشاراً وأكثرها استخداماً اليوم.

لغة جافا كغيرها من لغات البرمجة لاتخلو من العيوب، ويكمن اعتبار لغة جافا بطيئة نسبياً. إن السرعة ميزة مهمة، ولكن يجب التضحية ببعض المميزات لاكتساب مميزات أهم.

وهذا اول برنامج لنتعرف على محتويات برنامج جافا

```java
class first
{
    public static void main(String args[])
    {
        System.out.println(" My First program in Java ");

        }// end of main
    } // end of class
```

```
C:\WINDOWS\system32\cmd.exe
 My First program in Java
Press any key to continue . . .
```

يبداء برنامج جافا بالكلمة المحجوزة **class** يليها اسم البرنامج الذي اختاره المبرمج وهنا **first** ويجب حفظ الملف بنفس الاسم ويحتوي ال **class** على الدالة **public static void main(String args[])** ويبداء تنفيذ البرنامج من هذه الجملة

انواع البيانات في الجافا : –

الاعداد الصحيحة : –

النوع	طوله في الذاكرة
Byte	1 byte
short	2 byte
int	4 byte
long	8 byte

الاعداد الحقيقية : –

النوع	طوله في الذاكرة
Float	4 byte
Double	8 byte

النوع المنطقي : –

Boolean ويشمل القيم **true** او **false**

النوع **String** : –

هذا النوع شائع الاستخدام على الرغم من انه من انواع البيانات غير الاساسية ويستخدم لتعريف النصوص

المتغيرات : –

تعريف المتغيرات : –

```
int number1;
int number1 , number2;
double num;
boolean test;
char ch;
String text;
```

وضع قيمة للمتغير : –

```
number1 = 6;
num = 6.8;
text = "Sudan";
test = true;
ch = 'a';
```

وهذا يعني ان قيمة المتغير **number1** هي **6** و قيمة المتغير **test** هي **true** وهكذا لبقية المتغيرات

مثال : –

```java
class country
{
    public static void main(String args[])
    {
        String count;
        count = "sudan";
        System.out.println("country is : " + count);
        count = "Ethopia";
        System.out.println("country is : " + count);
        }
    }
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
country is : sudan
country is : Ethopia
Press any key to continue . . .
```

ربط سلاسل نصية :-

لربط السلاسل النصية نستخدم المعامل (+)

والمثال التالي يوضح ذلك

```java
class today
{
    public static void main(String args[])
    {

        String text = "today is : ";
        String day = "friday";
        String output = text + day;
        System.out.println(output);

        }
    }
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
today is : friday
Press any key to continue . . .
```

الدالة **print** و **println** تقوم هذه الدوال بعملية الطباعة على الشاشة ولكن الدالة **println** بعد الفراغ من الطباعة تنتقل الي سطر جديد

تعريف الثوابت : –

الثابت هو متغير لا يمكن تغير قيمته في البرنامج ولكننا فقط نقوم بتعريفه ووضع قيمة ابتدائية له لحظة التعريف، وتظل هذه القيمة ثابتة طوال البرنامج. تعريف الثوابت لا يختلف عن المتغيرات إلا في الكلمة المحجوزة **final** والتي نكتبها أمام التعريف لنستدل بها على أنه ثابت.

```java
final char plus = '+';
final double pi = 3.14;
```

العمليات الرياضية في الجافا : –

طريق الكتابة	العلامة	العملية
a + b	+	الجمع Addition
a − b	−	الطرح Subtraction
a * b	*	الضرب Multiplication
a/b	/	القسمة Division
a % b	%	باقي القسمة Modulus

العبارات المنطقية : –

قيمتها	معناها	العملية
صواب فقط إذا كان كل من x و y صواب	x and y	&&
خطأ فقط إذا كان كل من x و y خطأ	x or y	||
خطأ إذا كان z صواب، وصواب إذا كان z خطأ	not z	!

مثال : –

```java
class math
{
    public static void main(String args[])
    {
        int num1 = 3, num2 = 4;
        int sum = num1 + num2;
        System.out.println("Sum is : " + sum);

        }
    }
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
Sum is : 7
Press any key to continue . . .
```

قراءة البيانات من المستخدم : –

لقراءة البيانات من المستخدم نستخدم الكائن **BufferedReaded** الموجود بالحزمة **java.io**

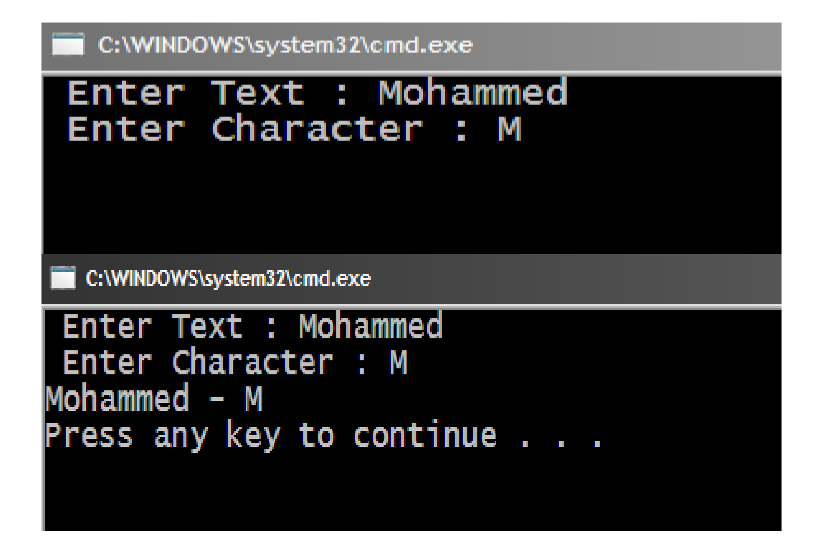
والبرنامج التالي يوضح ذلك

```java
import java.io.*;
// to use BufferedReader
class input
{
	public static void main(String args[])throws IOException
	{
	String text;
	char ch;
	BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
	System.out.print(" Enter Text : ");
	text = in.readLine();
	System.out.print(" Enter Character : ");
	ch = (char)in.read();
	System.out.println(text + " - " + ch);
	}
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
Enter Text :
```

```
C:\WINDOWS\system32\cmd.exe
Enter Text : Mohammed
```



طريقة اخرى لقراءة البيانات من المستخدم : –

نستخدم الكائن **in** الموجود في الحزمة **System** ولعمل ذلك نستخدم الفئة **Scanner** الموجودة في

import java.util.scanner

والمثال التالي يوضح ذلك

```
import java.util.Scanner;
// to import scanner class
class input
{
    public static void main(String args[])
    {
        Scanner in = new Scanner(System.in);
        System.out.print(" Enter number : ");
        int m = in.nextInt();
        System.out.println(" Number is : " + m);
    }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
 Enter number : 6
 Number is : 6
Press any key to continue . . .
```

ونلاحظ ان هذه الطريقة اسهل في الادخال

لادخال قيمة من النوع **char** نستخدم **in.nextChar();** و **in.nextString()** للسلاسل

عبارات المقارنة : –

أكبر من	>
أكبر من أو يساوي	>=
أصغر من	<
أصغر من أو يساوي	<=
يساوي	==
لا يساوي	!=

جمل الشرط : –

عبارة الشرط if : –

الصيغة العامة لعبارة **if**

```java
if(condition)
{
    statement ;
}
```

والمثال التالي يوضح ذلك

```java
// if statement program
import java.util.*;

class IfStatement
{
    public static void main(String args[])
    {
        int degree;
        Scanner input = new Scanner(System.in);
        System.out.print(" Enter your Degree : ");
        degree = input.nextInt();

        if(degree >= 50)
        {
            System.out.println(" Pass ");
        }

    }
}
```

الخرج من البرنامج

- القيمة المدخلة اكبر من **50**

```
C:\WINDOWS\system32\cmd.exe
 Enter your Degree : 89
 Pass
Press any key to continue . . .
```

- القيمة المدخلة اقل من 50

```
C:\WINDOWS\system32\cmd.exe
 Enter your Degree : 45
Press any key to continue . . .
```

-

العبارة if else : –

```
if(condition)
    statement1 ;
else
    statement2 ;
```

عندما تكون قيمة الشرط **condition** صواباً، يتم تنفيذ **statement1** وتجاهل **else** والعبارة التي تليها. وعندما يكون الشرط **condition** خطأ يتم تجاهل العبارة **statement1** وتنفيذ العبارة **statement2** . وكما في عبارة **if** ، إذا كان المطلوب تنفيذ أكثر من أمر واحد في حالة قيمة الشرط خطأ، توضع الأوامر بين قوسي بداية ونهاية.

الان سنقوم بتعديل المثال السابق

```java
// if statement program
import java.util.*;

class IfStatement
{
    public static void main(String args[])
    {
        int degree;
        Scanner input = new Scanner(System.in);
        System.out.print(" Enter your Degree : ");
        degree = input.nextInt();

        if(degree >= 50)
            System.out.println(" Pass ");
        else
            System.out.println(" Fail ");
    }
}
```

الخرج من البرنامج

عندما يدخل المستخدم قيمة اكبر من 50

```
C:\WINDOWS\system32\cmd.exe
 Enter your Degree : 78
 Pass
Press any key to continue . . .
```

وعندما يدخل قيمة اقل من 50

```
C:\WINDOWS\system32\cmd.exe
 Enter your Degree : 47
 Fail
Press any key to continue . . .
```

مثال : –

برنامج يحدد اذا كان العدد يقبل القسمة على 6

```
// program to known number is devisable by six
import java.util.*;
class test1
{
    public static void main(String args[])
    {
        Scanner in = new Scanner(System.in);
        int number1;
        System.out.print(" Enter number : ");
        number1 = in.nextInt();
        if(((number1%2)== 0)&&((number1%3)==0))
            System.out.println(" number is devisable by 6 ");
        else
            System.out.println(" number is not devisable by 6 ");
    }
}
```

الخرج من البرنامج

عندما ادخل المستخدم الرقم 24

```
C:\WINDOWS\system32\cmd.exe
 Enter number : 24
 number is devisable by 6
Press any key to continue . . .
```

عندما ادخل المستخدم الرقم 56

```
C:\WINDOWS\system32\cmd.exe
 Enter number : 56
 number is not devisable by 6
Press any key to continue . . .
```

ويمكن التعبير عن ال **if else** ب

max = (number1< numner2)?number1:number2;

المثال التالي يوضح ذلك

```java
import java.util.*;
class test2
{
    public static void main(String args[])
    {
        Scanner in = new Scanner(System.in);
        int number1 , number2;
        System.out.print(" Enter number 1 : ");
        number1 = in.nextInt();
        System.out.print(" Enter number 2 : ");
        number2 = in.nextInt();
        int max = (number1 > number2) ? number1 : number2;
        System.out.println(" Maximum number is " + max );
    }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
 Enter number 1 : 78
 Enter number 2 : 90
 Maximum number is 90
Press any key to continue . . .
```

العبارة **......if else if else** : –

```
if(condition)
    statement;
else if(condition)
    statment;
else
    statement;
```

العبارة switch : –

الصيغة العامة للعبارة **switch**

```
switch(variable)
{
    case value1:
        statement;
    break;
    case value2:
        statement;
    break;
    default:
        statement;
}
```

حيث **variable** هو اسم المتغير المطلوب إجراء الاختبارات على قيمته، ويشترط فيه أن يكون من النوع **int** أو **char** . **value2, value1** عبارة عن قيم يمكن أن يأخذها المتغير.

عند إجراء الاختبار على المتغير **variable** ، إذا ساوت قيمته أيّاً من القيم الموجودة بعد كلمة **case** ، يتم تنفيذ العبارة أو العبارات التالية حتى الوصول إلى نهاية **switch** أو العثور على الكلمة **break** ، والتي تقوم بإيقاف تنفيذ عبارات **case** التالية لعبارة **case**

التي تم تنفيذها. أما إذا احتوى المتغير على قيمة غير موجودة ضمن عبارات **case** ، عندئذ يتم تنفيذ العبارات التالية للكلمة المحجوزة **default** .

مثال : –

```
import java.util.*;
class switchstatement
{
    public static void main(String args[])
    {
        Scanner in = new Scanner(System.in);
        int digit;
        String number;
        System.out.print(" Enter number  : ");
        digit = in.nextInt();
        switch(digit)
        {
            case 1:
                number = "one";
            break;
            case 2:
                number = "two";
            break;
            case 3:
                number = "three";
            break;
            default:
                number = " number is " + digit;
        }
        System.out.println(number);
    }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
 Enter number  : 1
one
Press any key to continue . . .
```

```
C:\WINDOWS\system32\cmd.exe
 Enter number  : 4
 number is 4
Press any key to continue . . .
```

الحلقات التكرارية : –

في كثير من البرامج، نحتاج لتكرار تنفيذ جزئية معينة من البرنامج لعدد من المرات، مثلاً إذا كان البرنامج يقوم بقراءة أسماء **50** موظفاً، ليس من المنطقي أن نكتب **50** عبارة قراءة مختلفة. أو إذا كان البرنامج يطبع الأعداد من **1** إلى **1000**، فلا يمكن تصور برنامج يحتوي على **1000** عبارة طباعة، لأنه سيكون طويلاً جداً، وفي نفس الوقت يحتوي على مجموعة من العمليات المتشابهة، وهي عملية الطباعة.

من المتوقع أن نحتاج إلى تكرار تنفيذ العبارات في أغلب البرامج، وخاصة البرامج الكبيرة والأنظمة لأنها تتعامل مع مجموعات من البيانات. ففي نظام للمرتبات، يتم حساب المرتب لكل موظف على حده. أي تكرار عملية حساب المرتب بعدد الموظفين. وفي نظام بنكي، للبحث عن اسم عميل بواسطة رقم حسابه، يتم المرور على جميع عملاء البنك واختبار أرقام الحساب إلى أن نجده أو ينتهي العملاء. ولذلك نجد أن للتكرار أهمية كبرى يكاد لا يستغني عنها أي نظام.

توفر لغة **Java** ثلاث عبارات مختلفة للتكرار. سنتناولها بالتفصيل.

الحلقة while : –

تقوم الحلقة **while** بتكرار العبارات بداخلها مادامت قيمة الشرط **condition** هي **true**

الصيغة العامة للعبارة **while**

```
while(condition)
{
    Statement;
}
```

مثال : –

```
// use while statement
class whileloop
{
    public static void main(String args[])
    {
        int i = 0;
        while(i <= 5)
        {
            System.out.println(" i = " + i);
            i++;
        }
    }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
 i = 0
 i = 1
 i = 2
 i = 3
 i = 4
 i = 5
Press any key to continue . . .
```

الحلقة do while : –

هي شبيهة بحلقة **while** إلا أنه يتم اختبار شرطها في نهاية الحلقة. أي أنها تقوم بتنفيذ العبارات الموجودة بداخلها ثم اختبار قيمة الشرط لتحديد استمرارية تكرار عباراتها أو توقفها.

الصيغة العامة للحلقة **do while**

```
do
{
    statement;
}while(condition);
```

مثال : –

برنامج يقوم بطباع مضاعفات العدد **12** ال **100**

```java
// use do while statement
class dowhileloop
{
    public static void main(String args[])
    {
        int i = 12;
        do
        {
            System.out.println("   " + i);
            i += 12;
        }while(i <= 100);
    }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
   12
   24
   36
   48
   60
   72
   84
   96
Press any key to continue . . .
```

الحلقة for : –

عبارة أو حلقة **for** تقوم بتكرار تنفيذ التعليمة **statement** لعدد معلوم من المرات. هذا العدد المعلوم عبارة عن عدد القيم التي يأخذها عداد الحلقة **counter** . يأخذ العداد القيمة الابتدائية **initialValue** ويتم تنفيذ العبارة **statement** ، وبعد كل تنفيذ تزداد قيمة

المتغير **counter** حسب ما هو معرّف في **incrementExpression** حتى يصل إلى القيمة النهائية **finalValue** ، وعندها يتوقف التكرار.

الصيغ العامة للحلقة **for**

```
for(start;end;frequency)
    statement ;
```

مثال : –

```
// use for statement
class forloop
{
    public static void main(String args[])
    {
        for(int i = 0;i < 10; i+=2)
            System.out.println(" " + i);
    }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
0
2
4
6
8
Press any key to continue . . .
```

الحلقة **for** المتداخلة : –

في هذا المثال نستخدم حلقة **for** داخل حلقة **for** اخرى

```java
// use nested for statement
class forloop
{
    public static void main(String args[])
    {
        for(int i = 0; i < 5; i++)
        {
            for(int j = 0; j < 5; j++)
                System.out.print(" * ");
            System.out.println();
        }
    }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
 *  *  *  *  *
 *  *  *  *  *
 *  *  *  *  *
 *  *  *  *  *
 *  *  *  *  *
Press any key to continue . . .
```

ملاحظة حول الحلقات التكرارية : –

- عندما نعلم سلفاً عدد التكرارات التي ستنفذها الحلقة، الأفضل استخدام حلقة **for** .

- إذا كنا لا نعلم عدد التكرارات تحديداً، وخصوصاً إذا كان التكرار يعتمد على قيمة يقوم بإدخالها المستخدم، في هذه الحلقة يفضل استخدام **while** أو **do while** .
- إذا كنا نحتاج لعداد لمعرفة رقم التكرار أو استخدام قيمته في البرنامج يمكن استخدام حلقة **for** للاستفادة من عدادها، حيث أن قيمته تبين رقم التكرار.
- إذا كان من الممكن ألا يتم تنفيذ الحلقة أصلاً، فالأصح استخدام حلقة **while** ، أما إن كان تنفيذ الحلقة يكتمل للمرة الأولى في كل الأحوال، يتساوى حينها استخدام **while** و **do while** .
- عموماً عند استخدام لغة **Java** يمكن أن نعبر عن أي فكرة بها تكرار بأيٍ من العبارات التكرارية الثلاث التي توفرها اللغة، وبصورة سليمة وصحيحة، ولكننا دائماً نختار الحلقة الأمثل والأفضل والتي تجعل كتابة البرنامج أسهل وأقل تعقيداً، وتؤدي المطلوب بصورة أكفأ وذلك حسب خواص الحلقة وطبيعة البرنامج المطلوب.

المصفوفات **Arrays** : –

المصفوفة عبارة عن صف من البيانات ذات علاقة ببعضها من نفس نوع البيانات، يكون للمصفوفة اسم واحد وعدد من الحجرات توضع بها البيانات.

المصفوفات احادية البعد : –

الصيغة العامة لتعريف المصوفة احادية البعد

```
DataType[] VariableName = new DataType[Number];
```

او

```
DataType VariableName[] = new DataType[Number];
```

والمثال التالي يوضح تعريف مصفوفة من النوع **int**

```
int array[] = new int[5];
```

وضع قيم ابتدائية لعناصر المصفوفة : –

الشكل التالي يوضح كيفية وضع قيم اولية للمصفوفة **array**

```
array[0] = 1;
array[1] = 2;
array[2] = 3;
array[3] = 4;
array[4] = 5;
```

تعريف المصفوفة واعطاءها قيم اولية

```
int array[] = new int[]{1,2,3,4,5};
```

طباعة عنصر من المصفوفة

```java
System.out.println(array[3]);
```

لطباعة كل عناصر المصفوفة نستخدم حلقة **for**

```java
for(int i = 0; i < array.length; i++)
    System.out.print(" " + array[i]);
```

مثال على المصفوفات احدية البعد

```java
import java.util.Scanner;
class student
{
    public static void main(String args[])
    {
        Scanner in = new Scanner(System.in);
        String name[] = new String[3];
        int degree[] = new int[3];
        for(int i = 0;i < 3;i++)
        {
            System.out.print(" Enter Name : ");
            name[i] = in.next();
            System.out.print(" Enter Degree : ");
            degree[i] = in.nextInt();
        }
        System.out.println(" Name " + " *** " + " Degree ");
        for(int j = 0;j < name.length;j++)
        {
            System.out.println(name[j] + "   ***   " + degree[j]);
        }
    }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
 Enter Name : md
 Enter Degree : 78
 Enter Name : ea
 Enter Degree : 88
 Enter Name : mn
 Enter Degree : 77
 Name  ***  Degree
md  ***  78
ea  ***  88
mn  ***  77
Press any key to continue . . .
```

المصفوفات متعددة البعد : –

الصيغة العامة لتعريف مصفوفة متعددة البعد

```
int arr2[][] = new int[2][3];
```

تعريف ووضع قيم اولية لمصفوفة متعددة الابعاد

```
int arr2[][] = new int[][] {{1,2,3},{4,5,6}};
```

طباعة عنصر محدد من المصفوفة متعددة الابعاد

```
System.out.print(arr2[0][1]);
```

لطباعة كل عناصر المصفوفة نستخدم حلقات **for** متداخلة

```java
for(int i = 0;i < arr2.length;i++)
{
    for(int j = 0;j < arr2[i].length;j++)
    {
        System.out.print(arr2[i][j] + "   ");
    }
    System.out.println();
}
```

مثال على المصفوفة متعددة البعد

```java
import java.util.*;

class arraytest1
{
    public static void main(String args[])
    {
        int arr2[][] = new int[][] {{1,2,3},{4,5,6}};
        for(int i = 0;i < arr2.length;i++)
        {
            for(int j = 0;j < arr2[i].length;j++)
            {
                System.out.print(arr2[i][j] + "   ");
            }
            System.out.println();
        }
    }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
1   2   3
4   5   6
Press any key to continue . . .
```

الدوال في الجافا : -

الدالة هي مجموعة من التعليمات التي تؤدي وظيفة معينة، يتم نداؤها خلال البرنامج عند الحاجة بواسطة اسمها. في لغة **Java** تسمى الدوال **methods** . وهناك الكثير من الدوال الموجودة والمعرّفة أصلاً في لغة **Java** والتي يمكن أن نستخدمها عندما نريد. من اكثر دوال لغة **Java** التي استخدمناها خلال الأمثلة الدالة **print** والدالة **println** ، وهما دالتان الغرض منهما الطباعة على الشاشة.

تحتوي لغة **Java** على عدد هائل جداً من الدوال ذات الوظائف المختلفة في شتى المجالات، ولا يتسع المجال لذكرها، بل ولا يمكن حصر جميع الدوال في متناول اليد، ولكن يبحث المبرمج عن الدوال التي يحتاجها بناءً على مجالها. ولاستخدام هذه الدوال لا بده من معرفة طريقة كتابتها ونوع وعدد الوسائط التي تاخذها .

الدوال الجاهزة : -

دوال ال **class math** : -

الطريقة	وصف الطريقة	مثال
abs(x)	القيمة المطلقة لـ x.	Math.abs(6.2) → 6.2 Math.abs(-2.4) → 2.4
ceil(x)	تقرّب x إلى أقل عدد صحيح ليس اقل من x.	Math.ceil(5.1) → 6 Math.ceil(-5.1) → -5
floor(x)	تقرب x إلى أكبر عدد صحيح ليس أكبر من x.	Math.floor(5.1) → 5 Math.floor(-5.1) →-6
max(x,y)	أكبر قيمة من x و y.	Math.max(7,6) → 7
min(x,y)	أقل قيمة من x و y.	Math.min(-7,-8) → -8
pow(x,y)	x مرفوعة للأس y.	Math.pow(6,2) → 6^2 → 36
sqrt(x)	الجذر التربيعي لـ x.	Math.sqrt(9) → $\sqrt{9}$ → 3
random()	تكوّن رقم عشوائي بين الصفر والواحد.	Math.random() → 0.23121

هذه بعض الدوال الموجودة في الفئة math

الدوال الخاصة السلاسل : –

إيجاد طول السلسلة الرمزية:	
s.length()	ترجع الطريقة length() طول السلسلة الرمزية s.
عمليات المقارنة بين سلستين رمزيتين (ملاحظة: لاتستخدم == و !=).	
s.compareTo(t)	تقوم الطريقة بمقارنة السلسلة الرمزية s مع السلسلة الرمزية t وتعيد رقم سالب اذا كانت s اقل من t وتعيد صفر إذا كانت s تساوي t وتعيد رقم موجب إذا كانت s أكبر من t.
s.compareToIgnoreCase(t)	تعمل هذه الطريقة بنفس عمل الطريقة compareTo() ولكن مع اهمال حالة الحروف (صغيرة أم كبيرة).
s.equals(t)	تعيد true إذا كان s يساوي t.
s.equalsIgnoreCase(t)	تعمل هذه الطريقة بنفس عمل الطريقة equals() ولكن مع إهمال حالة الحروف (صغيرة أم كبيرة).
s.startsWith(t)	تعيد true إذا كان s يبدأ بالسلسلة الرمزية t.
s.startsWith(t, i)	تعيد true إذا كانت السلسلة الرمزية t موجودة في s بدءاً من الموقع i.
s.endsWith(t)	تعيد true إذا كان s تنتهي بـ t.

عمليات البحث: كل طرق ()indexOf تقوم بإرجاع 1- إذا كان العنصر المراد البحث عنه غير موجود، ويمكن للعنصر المراد البحث عنه أن يكون حرف أو سلسلة رمزية.	
`s.indexOf(t)`	ترجع موقع أول مكان توجد فيه t داخل السلسلة الرمزية s.
`s.indexOf(t, i)`	ترجع موقع أول مكان توجد فيه t داخل السلسلة الرمزية s بعد الموقع i.
`s.indexOf(c)`	ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s.
`s.indexOf(c, i)`	ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s بعد الموقع i.
`s.lastIndexOf(c)`	ترجع موقع آخر مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s.
`s.lastIndexOf(t)`	ترجع موقع آخر مكان توجد فيه السلسلة الرمزية t داخل السلسلة الرمزية s.

عمليات أخذ جزء من السلسلة الرمزية string.	
s.charAt(i)	ترجع الحرف الموجود في الموقع i داخل السلسلة الرمزية s.
s.substring(i)	ترجع جزء من السلسلة الرمزية s بدءاً من الموقع i وحتى النهاية.
s.substring(i, j)	ترجع جزء من السلسلة الرمزية s بدءاً من الموقع i وحتى الموقع j-1.
عمليات التعديل على السلسلة الرمزية string وإنشاء سلسلة رمزية جديدة.	
s.toLowerCase()	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية s بعد تحويل كل الحروف إلى حروف صغيرة.
s.toUpperCase()	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية s بعد تحويل كل الحروف إلى حروف كبيرة.
s.trim()	إنشاء سلسلة رمزية جديدة من السلسلة الرمزية s بعد الفارغ من البداية والنهاية.
s.replace(c1, c2)	إنشاء سلسلة رمزية جديدة من السلسلة الرمزية s بعد تبديل كل c1 بـ c2، وهما من نوع char.

عمليات أخرى على السلاسل الرمزية string.	
s.matches(regexStr)	ترجع هذه الطريقة true إذا كانت السلسلة الرمزية regexStr تطابق السلسلة الرمزية s كاملة.
s.replaceAll(regexStr, t)	إنشاء سلسلة رمزية string جديدة بعد تبديل كل regexStr بـ t.
s.replaceFirst(regexStr, t)	إنشاء سلسلة رمزية string جديدة بعد تبديل أول regexStr بـ t.
s.split(regexStr)	إنشاء مصفوفة تحتوي على أجزاء من السلسلة الرمزية s مقسمة حسب ظهور regexStr.
s.split(regexStr, count)	كما في الطريقة split(regexStr) لكن مع تحديد عدد مرات التقسيم.

الدوال المعرفة بواسطة المستخدم : –

الشكل العام لتعريف الدالة

```
access static return_type method_name(parameters)
{
     statement1;
     statement2;
     .
     .
     .
}
```

Access محدد الوصول ويكون **public** او **private** او **protected**

Static تستخدم لتعريف الدالة ليتم استخدامها داخل الصنف الذي عرفت فيه فقط

.

Return_type يحدد نوع القيم التي تعيدها الدالة

Method_name اسم الدالة

Parameters هي المعاملات . وعند تعريف الدالة تسمى هذه المعاملات بالمعاملات الشكلية (**Formal Parameters**) وعند استدعاء الداله تسمى بالمعاملات الفعلية (**Actual Parameters**)

مثال : -

```
public static void university()
{
    System.out.println(" Alzaim Alazhari University ");
}
```

دالة لا تعيد قيمة ولاتحمل وسائط هذه الدالة تقوم بطباعة النص **Alzaim Alazhari University**

اشكال الدوال :-

- دالة لا تاخذ وسائط ولا تعيد قيمة
- دالة تاخذ وسائط ولا تعيد قيمة
- دالة تاخذ وسائط وتعيد قيمة

• دالة لا تاخذ وسائط وتعيد قيمة

مثال يوضح اشكال الدوال :-

```
public static void method1()
{
    System.out.println("method 1");
}
public static void method2(String method2)
{
    method2 = "method 2";
    System.out.println(method2);
}
public static int method3(int x)
{
    return x * x;
}
public static int method4()
{
    int x,pi = 3.14;
    return x * pi;
}
```

استدعاء الدوال : –

يتم استدعاء الدالة باسمها كما في الشكل التالي

```
university();
```

مثال : –

```java
class method
{
     static int sum(int a, int b)
     {
          return a * b ;
     }
     public static void main(String[] args)
     {

          System.out.println(" Sumation is " + sum(3,4));
     }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
 Sumation is 12
Press any key to continue . . .
```

النداء الذاتي : –

هو ان تقوم الدالة باستدعاء نفسها بنفسها . البرنامج التالي يقوم بايجاد مضروب العدد **n** باستخدام النداء الذاتي .

```java
import java.util.*;
class recursion
{
    static int fact(int a)
    {
        if(a == 1 || a == 0)
            return 1;
        else
            return a * fact(a - 1);
    }
    public static void main(String args[])
    {
        Scanner in = new Scanner(System.in);
        int num;
        System.out.print(" Enter Number : ");
        num = in.nextInt();
        System.out.println(" Factorial : " + fact(num));
    }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
 Enter Number : 5
 Factorial : 120
Press any key to continue . . .
```

ملاحظة : –

- عند استخدام النداء الذاتي يجب الانتباه إلى ضرورة وجود شرط معين لإيقاف النداء الذاتي، وإلا ستتواصل النداءات لعدد لانهائي من المرات، وعندها لا يتوقف البرنامج عن التنفيذ .

- عند استخدام النداء الذاتي يجب الاحتراس والتأكد من وجود شرط توقف النداءات. لكن الأفضل استبداله بالحلقات لأن تنفيذ البرنامج بالنداء الذاتي يستغرق زمناً أطول في التنفيذ ويستهلك ذاكرة أكبر من تنفيذ نفس البرنامج باستخدام الحلقات.

تحميل الدوال بشكل زائد : –

تتم عملية تحميل الدوال بشكل زائد عندما تكون هناك اكثر من دالة تحمل نفس الاسم في نفس الفئة ويتم التميز بين هذه الدوال من خلال عدد المعاملات التي تحملها وانواعها

مثال : –

```java
class method
{
     static int sum(int a, int b)
     {
          return a + b ;
     }
     static int sum(int a,int b,int c)
     {
          return a + b + c;
     }
     static double sum(double a,double b)
     {
          return a + b;
     }
     public static void main(String[] args)
     {

          System.out.println(" Sumation is " + sum(3,4));
          System.out.println(" Sumation is " + sum(3,4,3));
          System.out.println(" Sumation is " + sum(3.2,4.3));
     }
}
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
 Sumation is 7
 Sumation is 10
 Sumation is 7.5
Press any key to continue . . .
```

المراجع : -

- Java how to program 7th edition
- البرمجة بلغة جافا – جامعة السودان المفتوحة

الفهرس